Universidad Zaragoza
1542

Trabajo Fin de Grado

Título del trabajo: Plataforma de computación perimetral para adquisición y procesado de vídeo y señales fisiológicas

English tittle: Edge computing platform for the recording and analysis of video and electrophysiological data

Autor/es

Pablo Noel Carreras Aguerri

Director/es

Darío Suárez Gracia
Luis Montesano del Campo

Ingeniería Informática

Escuela de Ingeniería y Arquitectura de Zaragoza
2020

Plataforma de computación perimetral para adquisición y procesado de vídeo y señales fisiológicas

Pablo Noel Carreras Aguerri

23 de septiembre de 2020

Resumen

Este proyecto se lleva a cabo en colaboración con la empresa start up de la Universidad de Zaragoza Bitbrain, especializada en la neurotecnología y el despliegue de dispositivos facilitadores de la vida de las personas. Para ayudar en este propósito, este TFG tiene como objetivo llevar a cabo un proyecto multidisciplinar dentro de la informática en el campo de neurotecnología que desarrolle un prototipo de estos dispositivos. El prototipo realizado consiste en un sistema capaz de grabar, procesar y hacer un stream de tanto vídeo como señales fisiológicas en tiempo real. Como dificultad adicional, Bitbrain no disponía de dispositivos portátiles que realizaran captación y procesado al mismo tiempo y no se disponía de mucha experiencia con el tipo de herramientas portátiles de procesado que se han usado en este TFG. El desarrollo parte de un conjunto de 3 piezas de hardware, una cámara de intel (RealSense D435i), una placa Jetson Nano de Nvidia y un conversor Analógico-Digital resarrollado por Bitbrain. Estos tres elementos se fusionan por medio de programas en c, c++, python, html y javascript para conseguir un prototipo que es capaz de capturar y procesar información de dos tipos de sensores simultáneamente y además, ofrecer una plataforma en la red local que permita no solo visualizar los datos, sino controlar el dispositivo y descargar los datos de los experimentos. El prototipo es plenamente funcional, demostrando que los elementos son compatibles y los objetivos alcanzables, sino que también se aportan experimentos que muestran el rendimiento y la eficiencia del sistema que dejan a la vista como la Jetson Nano es un producto muy válido para estos casos de uso. Pese a todo, como el tiempo de desarrollo de un TFG es limitado, el proyecto es muy abierto y se dejan puertas abiertas para trabajo futuro en caso de continuar expandiéndolo. Estas son desde necesidades de mejora del sistema hasta implementaciones de nuevas funcionalidades.

Índice

1. Introducción

1.1. Motivación

Hoy en día, el campo de la neurotecnología se encuentra en pleno desarrollo y van saliendo al mercado numerosos dispositivos de captación de señales fisiológicas como las señales electroencefalográficas (EEG). En muchas situaciones, estas señales EEG son muy difíciles de analizar debido a la falta de contexto visual, auditivo, olfativo, tactil... que es necesario para conocer los estímulos que generan la señales. Una manera de solucionar este problema es la fusión de EEG con otros sensores para capturar conjuntamente todos los sensores y posteriormente analizar y asociar los eventos de los distintos sensores. Este concepto parece sencillo, pero tiene una gran complejidad para llevarse a cabo, ya que suele necesitarse una potencia computacional complicada de conseguir en un dispositivo portátil. En muchos experimentos, el sujeto necesita moverse llevando consigo el dispositivo captador y analizador. De esta idea surge este TFG en colaboración con la empresa start up de la Universidad de Zaragoza especializada en neurotecnología Bitbrain, especialmente interesada en prototipos que faciliten la vida de las personas, para la creación de un prototipo de estos dispositivos que fusione procesamiento de visión por computador a la captación de EEG. Además, esta empresa no cuenta todavía con ningún prototipo de este estilo, por el momento se han encargado de la captación y del procesado por separado. Esto hace que el proyecto suponga un reto aun mayor, ya que no se cuenta con modelos ni experiencias previas para guiar el desarrollo. Para esto se pretende trabajar con los distintos departamentos de Bitbrain para cubrir la variedad de áreas de la informática que cubre el proyecto.

1.2. Contexto

En el mercado podemos encontrar sobre todo dispositivos especializados en una de las dos tareas mencionadas. O bien tareas de procesamiento de datos, ejecución de redes neuronales, etc, o bien tareas de captación y almacenamiento en la nube de información fisiológica [1, 2]. También hay otros que van más allá y capturan el contexto por medio de cámaras y otros sensores. Sin embargo, estos se suelen encargar únicamente de la captación, y el procesado se realiza a posteriori. Pocos prototipos intentan realizar un procesamiento de datos en tiempo real en el mismo dispositivo de captación, cuando las herramientas de procesado de hoy en día podrían tener capacidad computacional suficiente para ejecutar ambas de forma concurrente. Es por eso, que haciendo uso de una plataforma hardware especializado en el procesamiento de datos, se pretende elaborar un dispositivo que sea capaz de captar, procesar, guardar y hacer un stream simultáneamente de vídeo y señales fisiológicas. Además de realizar esta tarea, se pretende que el dispositivo sea extensible y permita añadir más procesados o elementos nuevos a su estado al final del proyecto y que cumpla con unos requisitos de usabilidad. Para esto último, se pretende que el dispositivo se pueda controlar por medio de una API accesible vía web o mediante consulta directa y permita visualizar los datos en una web ofrecida por su propio servidor web local.

1.3. Objetivos

El objetivo del proyecto es desarrollar un prototipo de un sistema de procesado embarcado y de señales fisiológica (EEG) y de videio que permita su adquisición sincronizada, el procesamiento en tiempo real y la comunicación de los datos adquiridos y/o de los resultados. Para conseguir este objetivo, se han identificado las siguiente tareas que conforman el grueso del trabajo realizado durante el TFG:

1. Buscar y seleccionar la plataforma hardware.
2. Diseñar sistemas que sean capaces de captar, guardar y hacer un streaming tanto de vídeo como de EEG.
3. Crear y desplegar en el arranque el servidor web local.
4. Procesar los datos (vídeo y EEG) en su camino desde los programas de captación hasta la página web donde se visualizan.
5. Realizar pruebas de rendimiento del sistema para comprobar que el prototipo funciona y demostrar que la idea del proyecto es factible.
6. Realizar un experimento simulando un caso de uso real.

1.4. Alcance

El proyecto parte del hardware necesario para adquisición (plataforma de procesamiento, cámara y lector de EEG) y el software para el procesamiento (redes neuronales ya entrenadas para procesamiento de vídeo). A partir de estos elementos, se realiza un trabajo multidisciplinar dentro de la informática que abarca las siguiente tareas y ámbitos.

1. Creación de scripts y configuración de Linux para mejorar prestaciones, automatizar tareas en el arranque, etc.
2. Ampliación y uso de conocimientos sobre streaming en tiempo real de vídeo.
3. Desarrollo de programas en bajo nivel que comuniquen los dispositivos hardware a conectar entre si.
4. Creación y configuración de un servidor web con servicios de API y web locales.
5. Comunicación de procesos concurrentes por medio de sockets y websockets.
6. Creación de programas a más alto nivel que realicen procesamiento de señales fisiológicas.
7. Análisis del rendimiento del sistema por medio de contadores hardware y la herramienta `perf`, entre otras herramientas.

Estas tareas se han realizado en dos periodos: un primer periodo de prácticas en empresa en Bitbrain y un segundo periodo de trabajo del TFG en la universidad. En el primero se cubrieron parte de los objetivos del 1 al 3 abordando sobre todo la parte de comunicación entre el hardware de Bitbrain y la placa de procesamiento. Esta distribución surgió para tener acceso tanto al asesoramiento de de la empresa como a sus herramientas (como por ejemplo un osciloscopio) y desarrollar un servicio web según sus estándares durante las prácticas. Y en la segunda, se cubren el resto de objetivos al mismo tiempo que se cubren algunos errores que se acarrean de las prácticas.

2. Arquitectura del Sistema

2.1. Descripción global

La captación y el procesado de señales biomédicas y de imágenes requiere de un conjunto de sub-sistemas que lleven a cabo cada una de las tareas necesarias desde la captación de las señales en los sensores hasta su análisis y visualización final. Este se observa en la figura 1 y lo primero que se destaca a primera vista, es que el sistema se encuentra claramente compuesto por tres elementos principales:

- Conversor analógico-digital de Bitbrain
- Cámara de intel Realsense D435i
- Placa Jetson Nano

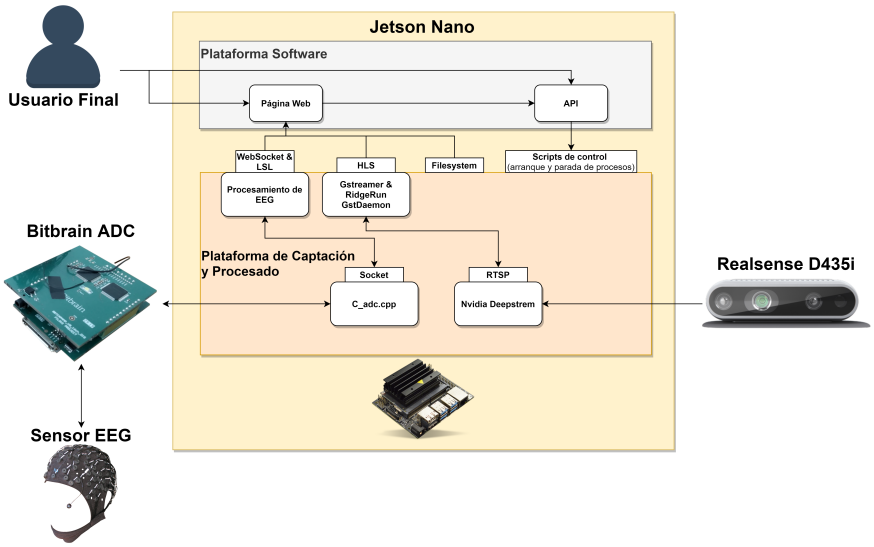

Figura 1: Diagrama de arquitectura del sistema final del proyecto

Es dentro de esta última en la que se desarrolla el contenido del proyecto para fusionar los elementos y ofrecer un servicio final al usuario. Se pretende así crear un conjunto de elementos de lectura y control de los que el usuario pueda hacer un uso delimitado a través de, o bien una página web simple e intuitiva,

o bien de peticiones directas a una API. La segunda opción permite un poco más de libertad para casos como por ejemplo, crear únicamente envíos de datos de EEG a una dirección IP e ignorar el procesamiento de vídeo (en la página web ambos procesamientos de EEG y vídeo funcionan juntos en bloque). Así pues, el software desarrollado en el proyecto dentro de la Jetson se clasifica de la siguiente manera:

- **Plataforma de captación y procesado**: consta de los elementos internos empleados en el proceso de obtener los datos de los elementos externos (vídeo y EEG), que además llevan a cabo los procesamientos necesarios y dejan el resultado final de forma accesible a la siguiente capa de servicios que permite su visualización.
  - **Control del *Analog Digital Converter* (ADC)**: parte encargada de obtener datos de EEG, hacer un primer procesado de decodificación y ponerlos a disposición de quien los solicite por medio de sockets.
  - **Nvidia Deepstream**: software de Nvidia creado especialmente para la captación, procesamiento y streaming de vídeo.
  - **Procesamiento de EEG**: Grupo de programas que se conectan al socket del controlador del ADC para aplicar procesamientos más sofisticados, hacer streamings en otros formatos, etc
  - **Gstreamer**: Programa de streaming de vídeo que se usa para captar el vídeo ya procesado, enviarlo a la web, guardar una copia local, guardar *screanshots*, etc.
- **Plataforma de servicios**: capa de interacción con el usuario final a través de la cual puede tener acceso a todos lo servicios por medio de las peticiones de la API, las cuales paran o ponen en marcha los distintos procesos de la placa.
  - **Api**: elemento que permite al usuario controlar la placa ejecutando los scripts que la capa inferior pone a disposición del usuario.
  - **Web**: página que de forma simple pone a disposición del usuario las funciones de la API y permite visualizar tanto el vídeo como la señal EEG.

2.2. Metodología

Previo a la descripción de cada sub-sistema se necesitan introducir las herramientas principales (asi como sus versiones) que se han empleado para su desarrollo.

2.2.1. Ubuntu

Como sistema operativo sobre el que arranca la placa, se ha utilizado la imagen del sistema que aporta Nvidia en su página [3]. Este es un Ubuntu que incluye software específico de Nvidia sobre todo para el uso de GPU, como por ejemplo CUDA.
Para el correcto funcionamiento del sistema, es necesario que sobre este sistema se pueda aplicar el parche *preempt_rt* (Real-Time Linux [4]). Este parche modifica el scheduling del kernel para que de mayor prioridad a las interrupciones hardware.

	Versión
Dispositivo	NVIDIA Jetson TX1
JetPack	L4T 32.3.1 [JetPack 4.3]
Ubuntu	Ubuntu 18.04.4 LTS
Kernel	4.9.140-rt93
CUDA	10.0.326

Cuadro 1: Conjunto de versiones del sistema instalado sobre la Jetson

Para el caso de uso sobre el que se trabaja en el proyecto, es fundamental que las interrupciones hardware producidas por el ADC de atiendan en tiempo real en el momento exacto en que se producen. Por esto, es importante que la versión del sistema sea superior a la versión *JetPack 4.3 - L4T R32.3.1*, ya que anteriores no son compatibles con este parche.

2.2.2. Apache

Por simplicidad y capacidades, es el software que se ha utilizado para el despliegue del servidor. Los tres servicios de los que se habla en la *Plataforma software* se configuran en un solo fichero de configuración de apache (este está en la ruta predeterminada para estos ficheros /etc/apache2/sites-enabled/). Adicionalmente se necesita también el paquete *libapache2-mod-wsgi* para poder desplegar las aplicaciones desarrolladas en flask.

2.2.3. RealSense

Lo único necesario para empezar a trabajar con la cámara de Intel, es descargar e instalar los drivers y demás software del paquete *librealsense* [5].

2.2.4. Gstreamer

Se trata de una de las herramientas más utilizadas para la transmisión de vídeo en Linux. Resumiendo su funcionalidad, esta permite captar vídeo de varios tipos de fuentes (cámaras, archivos de vídeo, etc) y transmitirlo por medio de un sistema de *pipelines*. El video parte de una fuente (*source*) y tiene que acabar en un extremo denominado *sink* que puede ser desde un guardado de la transmisión a un fichero hasta un formato en otro streaming para captarlo por otras instancias de Gstreamer (Figura 2). En el camino de un extremo a otro es donde se puede introducir cualquier tipo de procesamiento del contenido de la *pipeline*.

Figura 2: Simplificación de una instancia de Gstreamer

A día de hoy, no solo existen una gran cantidad de paquetes para hacer todo tipo tipo de procesamientos y streamings en distintos formatos, sino que ofrece soporte para crear filtros personalizados en distintos lenguajes como C o python. Es esta versatilidad la que lo hace muy apropiado para el proyecto. Se puede obtener más información al respecto de esta elección en el Anexo A, y sobre Gstreamer en su página web [6]

Software que trabaja sobre Gstreamer: para el proyecto, se han usado paquetes de todos los conjuntos denominados *good*, *bad* y *ugly*. Además, para mayor control y comodidad del sistema se han usado dos soportes *opensource* de la empresa RidgeRun [7]: GstDaemon y Interpipe [8, 9]. En su parte correspondiente en el procesado de vídeo se puede ver en uso concreto que se ha hecho de ellos.

2.2.5. Bibliotecas C++

Sobre todo para la Plataforma Software, se han usado bibliotecas de C++ para simplificar tareas concretas sobre comunicación:

- Websocketpp [10]
- Json for Modern C++ [11]

3. Platafroma de Captación y Procesado

Esta parte del sistema es la encargada de todas las tareas de extracción, procesamiento y streaming de datos e incluye el software desarrollado junto con las configuraciones y versiones del sistema operativo, programas usados de terceros, etc. Adicionalmente, aunque forman también parte de esta capa, los programas desarrollados para el procesamiento de datos se tratan en sus respectivas secciones.

3.1. Control del ADC

Para esta labor, se debe conectar la Jetson Nano con el conversor analógico-digital (ADC) de Bitbrain. Este tiene registros de datos divididos en bloques de 8 (Como es un ADC de 16 canales hay bloques A y B) y está preparado para recibir las señales de control del dispositivo a través de la cabecera de pines j41. El reparto de estos en la Jetson se puede observar en la figura 14 del anexo D, y sus funciones son las siguientes:

1. **Start**: arranca el sistema una vez configurado
2. **Reset**: resetea el valor de los registros
3. **PWDN**: activa o desctiva la placa
4. **DR**: indica la disponibilidad de datos en el ADC cuando su valor es 1
5. **CS1_A**: pin donde la jetson debe indicar con el valor 1 si esta realizando una modificación sobre los registros del Bloque A
6. **CS1_B**: pin donde la jetson debe indicar con el valor 1 si esta realizando una modificación cobre los registros del Bloque B

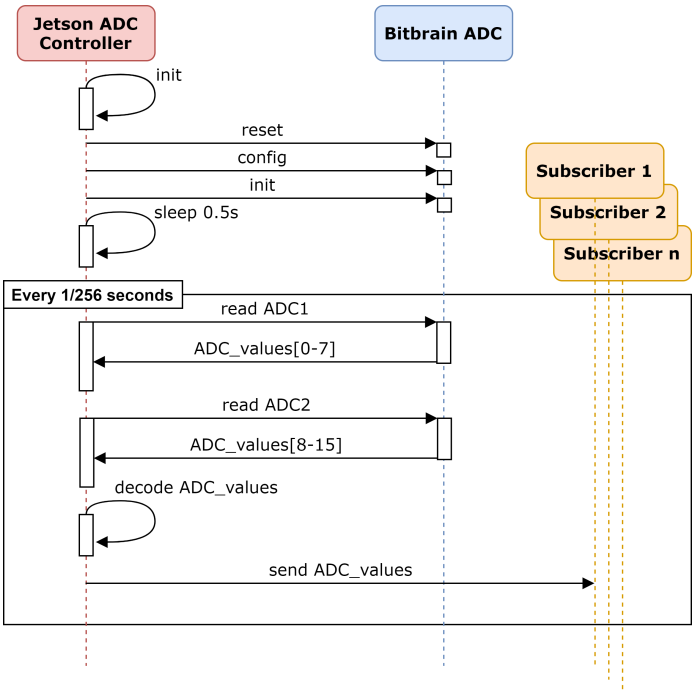

Figura 3: Diagrama de secuencia correspondiente al programa principal que se comunica con el ADC

El SPI, Serial Peripheral Interface, es el bus empleado para comunicar el ADC con la Jetson Nano y se puede configurar para trabajar desde la zona de usuario a través de lo que se conoce como Spidev [12], este usa los pines que del spi1 indicados que se pueden ver en la figura 14. A partir de la versión *JetPack 4.3 - L4T R32.3.1*, esta cabecera de pines se puede configurar de forma muy sencilla por medio de scripts puestos a disposición del usuario por Nvidia.
Con el hardware preparado, para llevar a cabo esta tarea, se necesita de un programa que tenga un buen control sobre las interrupciones y sea eficiente en tiempo, por ello, se desarrolla sobre c++. Tal y como se muestra en el diagrama de la figura 3, el programa funciona en bucle siguiendo el siguiente comportamiento mientras no se pare mediante la señal controlada SIGUSR1:

1. Inicialización de todos los componentes propios del programa: fichero de escritura para guardar una copia de los datos, estructuras de datos, sockets, spi y ADC (tras la configuración del ADC hace falta na espera de medio segundo).

2. A una frecuencia de 256Hz se leen los 16 canales de EEG en dos lecturas de 8 canales. Estos datos en crudo necesitan un ligero ajuste de *decode* y una vez listos, estos se guardan en un fichero en formato csv y se envían a todos los clientes subscritos al sistema *publish-subscribe*.

Tanto para escribir como escribir sobre los registos del ADC y ya sea en los registros de datos o los de configuración, se sigue la comunicación descrita por la figura 4.

Por otro lado, también se implementa un programa para llevar a cabo tanto lecturas como escrituras mediante los pines del SPI. Este solo replica un modelo estándar de comunicación por medio de SPI. Este se puede encontrar en diversidad explicaciones de internet como la mostrada en [13].

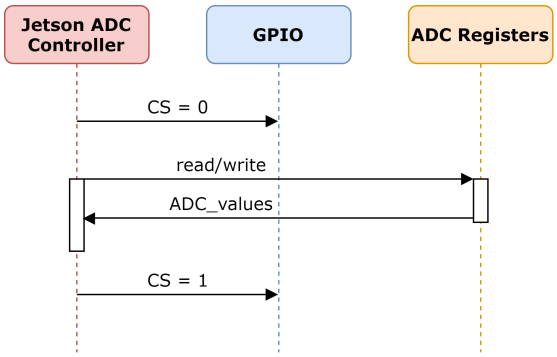

Figura 4: Diagrama de secuencia de como el programa de control del ADC ejecuta una lectura o escritura sobre los registros del ADC

3.1.1. Restricciones temporales

El principal problema a afrontar es la restricción temporal de leer a 256Hz. Esto implica que hay una ventana de 1/256 segundos para realizar dos lecturas, procesar los datos, guardarlos en un fichero y enviarlos a los clientes. A nivel de implementación, sobre C++ se consigue un tiempo constante de entre 1 y 2 milisegundos. Sin embargo, tal y como se comenta en al principio de la sección sobre el sistema operativo, si no se aplica el parche *preempt_rt* (disponible desde Enero de 2020 a mitad del desarrollo del proyecto), aparecen ciertas anomalias temporales que imposibilitan este trabajo:

- **Frecuencia**: si no se atienden las interrupciones en el momento en que se producen, no se puede dejar que el programa repose mientras se espera al siguiente ciclo. Esto ocurre tanto en interrupciones del propio programa (con funciones sleep), como en interrupciones de Linux (programación de alarmas de Linux) e interrupciones hardware. Esto hace que el programa deba permanecer activo consumiendo recursos para evitar bajadas en la frecuencia de lectura.
- **Lectura**: cuando se termina una lectura por el SPI, el ADC avisa del fin de esta transacción por medio de una interrupción hardware, que no se atiende en el momento y produce un descenso en la frecuencia de hasta 10 ciclos.

Estos retrasos temporales afectan a la frecuencia de lectura, y también a la precisión de los datos extraídos del ADC. Estos problemas únicamente se pueden

solventar con la actualización del sistema y a la aplicación del parche. Sin este, se pueden observar retrasos a la hora de atender las interrupciones a nivel de cpu. Sobre esta problemática se ha trabajado por medio de `perf`, una herramienta de análisis de prestaciones que permite entre otras cosas ver la actividad de la cpu durante la ejecución de un programa y `traceshark` [14], con el que se pueden visualizar los datos extraídos. Se explican y muestran resultados de estos experimentos en el Anexo C.

3.1.2. Patrón Publish-subscribe

Así pues, el objetivo de este programa es guardar los datos extraídos y servirlos en tiempo real a cualquier otro programa que quiera hacer uso de ellos. Para esto, se crea un servidor por sockets que envía los datos a todo aquel que los solicite. De esta forma, los programas de procesado tienen que enviar un primer mensaje para suscribirse y empezar a recibir los datos. Para evitar saturación en caso de que uno de estos programas se ralentice, el programa debe enviar una solicitud de datos entre cada lectura. Es decir, si el programa que los recibe no es capaz de procesar los datos en menos de 1/256 segundos y volver a pedir los datos, se saltará un bloque de datos, pero no se le saturará la la entrada. Si un programa lleva muchos envíos sin solicitar datos, se le puede borrar de la lista de suscriptores.

3.2. Comunicación entre capas

Para poder controlar la ejecución de esta capa desde la plataforma de servicios, se disponen una serie de scripts que se ejecutan desde la API con el paquete *os* de python. Estos son los siguiente:

- **init_stream.sh y close_stream.sh**: pone en marcha el controlador del ADC e inicia el sistema de *pipelipelines* de GstDaemon (explicado en el procesamiento de vídeo). Estos son los dos programas base sobre los que se despliegan el resto de funcionalidades. Así mismo, se puede acabar con estos procesos con el segundo script que busca los PIDs y envía una señal de parada.
- **EEGfilter_start.sh y EEGfilter_stop.sh**: Inicia o para como en el caso anterior programas de procesamiento de EEG.
- **pylslstream_start.sh y pylslstream_stop.sh**: empieza o para de transmitir los datos crudos de EEG en formato lsl.
- **recording_start.sh y recording_stop.sh**: empieza a para de guardar en un archivo local el streaming.
- **hlsstream_start.sh y hlsstream_stop.sh**: empieza o para de realizar una transmisión del streaming en formato HLS.
- **take_snapshot.sh**: función de prueba que guarda localmente una captura de ese instante del straming.

4. Plataforma de Servicios

En este bloque, se incluyen los servicios web que se ofrecen al usuario para que pueda hacer control del prototipo. Para esto se han desarrollado una API, que permite ejercer acciones sobre la placa delimitadas por peticiones y un servidor web local, que permite a cualquier dispositivo conectado a la misma red local conectarse para controlar la placa mediante un modelo más sencillo de formularios y visualizar en tiempo real tanto vídeo como EEG. Si sumamos a esto un servicio de directorios virtual para descargar los datos grabados, el *hostname* de la Jetson se encuentra establecido para poder acceder localmente a la placa a través de las siguientes direcciones:

1. `http://jetsonnano-desktop.local/WebController`
2. `http://jetsonnano-desktop.local/Downloads`
3. `http://jetsonnano-desktop.local/apirest`

Ya que para ambos ambos elementos se pretende desarrollar un sistema simple y rápido, se ha optado por el desarrollo en python por medio de flask, un paquete de python que permite desarrollar servicios web de forma rápida.

4.1. API REST

Como se ha planteado en un inicio, esta API es la única forma de manipular la Jetson de forma remota. Se trata de una API simple en python, que ejecuta los scripts de la capa inferior disponibles en las siguientes peticiones:

- `http://jetsonnano-desktop.local/apirest`
  - **/Recording**: control de la gravación de vídeo.
    - action : {start, stop}
  - **/StreamingHLS**: control del streaming en formato HLS.
    - action : {start, stop}
  - **/Snapshot**: funcionalidad para tomar imágenes instantaneas.
    - action : {take}
  - **/EEGfilterStart**: permite esmpezar un streaming de filtrado de EEG.
    - filter : {"filtro de EEG"}
  - **/EEGfilterStop**: para el streaming del filtrado.
  - **/InitCamera**: Inicia la cámara.
    - source : {"fuentede_vídeo"}
  - **/InitPylslstream**: inicia un streaming de EEG en formato LSL.
  - **/EndAll**: Para el funcionamiento de la cámara
  - **/IsCameraAlive**: Devuelve {200 : True} o {200 : False} dependiendo de si la cámara está siendo detectada en la placa.

A estos servicios se puede acceder tanto desde la página web, donde se ejecutan automáticamente como desde cualquier programa que permita realizar peticiones como por ejemplo *curl*. Los siguiente son ejemplos de peticiones realizadas con este:

- *curl http://jetsonnano-desktop.local/apirest/IsCameraAlive -X PUT*
- *curl http://jetsonnanodesktop.local/apirest/Recording -d "action=start" -X PUT*
- *curl http://jetsonnanodesktop.local/apirest/EEGfilterStart -d "filter=impedances" -X PUT*

4.2. Servidor Web

El único objetivo de este servicio es aportar una interfaz al usuario para usar el dispositivo. A este usuario, se le presenta un formulario que permite seleccionar el tipo de procesado de vídeo y realizar tanto un stream lsl local como un stream rtp a una lista de direcciones IP. Una vez confirmados los ajustes, se redirige a la página que se muestra en la figura 5 donde visualizar el streaming en formato HLS y se puede seleccionar el tipo de información de EEG que se quiera visualizar sobre una gráfica en tiempo real. Para parar la sesión, se dispone de un botón en la parte inferior de la página.

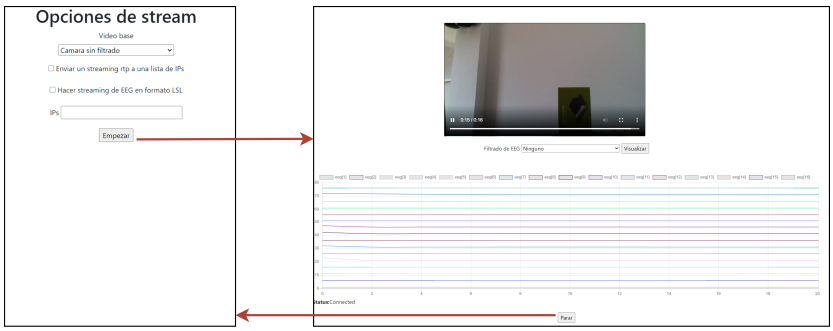

Figura 5: Diagrama de navegación de la página web local

En cuestiones de implementación, Los formularios se hacen de forma simple sobre flask al igual que las peticiones a la api con la librería *requests*. De la misma forma, en el HTML, es donde se emplean, tanto la librería mencionada de websockets para la recepción de EEG, como otra librería de javascript para visualizar el streaming HLS cuyo nombre es hls.js [15].

5. Procesamiento de EEG

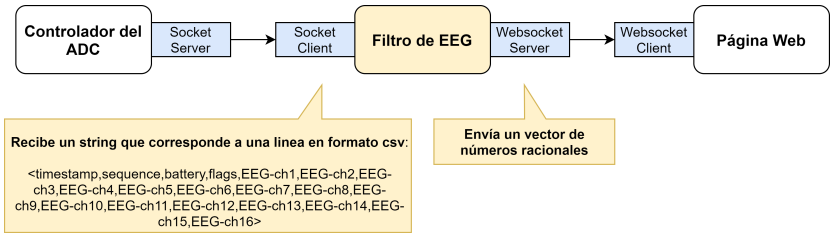

Figura 6: Ejemplificación de la entrada y salida del filtro de EEG

Si recordamos la figura 1, esta parte hace la labor de tubería entre el programa que capta la señal y lo que se visualiza en la web (figura 6). El programa desarrollado para esta labor `EEG_filter.cpp` ofrece una estructura genérica para el desarrollo de más filtros. Como este se inicia con un string que indica el tipo de procesado a realizar, para cada nuevo procesado que se añada, solo habría que añadir el tipo y una función a la que llamar. Para crear esta función, hay que ser conscientes del ciclo que lleva a cabo el programa. En primer lugar, cada 1/256 segundos, a través de un socket recibe una cadena csv con el mismo formato con el que se almacena la señal:

> timestamp , sequence , battery , flags ,EEG–ch1 ,EEG–ch2 ,
> EEG–ch3 ,EEG–ch4 ,EEG–ch5 ,EEG–ch6 ,EEG–ch7 ,EEG–ch8 ,
> EEG–ch9 ,EEG–ch10 ,EEG–ch11 ,EEG–ch12 ,EEG–ch13 ,
> EEG–ch14 ,EEG–ch15 ,EEG–ch16

Esta se divide y se extraen los 16 canales de EEG. Tras la división, se procesa la señal y se declara por medio de un booleano si se genera un nuevo vector que enviar de cualquier tamaño superior a uno (esta sería la labor de la función de un nuevo procesado). En los filtros ya creados, esto se suele hacer una vez cada segundo, por seguir con el sistema que se llevaba en el primer filtro, no estresar el visor web y conseguir una mejor visualización. Así finalmente, este vector se envía por medio de un servidor de *websockets* instanciado de la librería previamente mencionada websocketpp [10]. Para esta primera versión del software, se encuentran implementados los siguientes filtros:

1. **no_filter**: simplemente se reenvía la información cruda de los 16 canales. Esta es la opción que se aplica si se llama al programa con un filtro desconocido.

2. **impedances**: se trata de un filtro de impedancias, es muy simple y rápido de aplicar. Con esta información se puede comprobar si el sensor de EEG está bien colocado sobre el sujeto de pruebas buscando el valor más bajo posible para la impedancia de cada canal.

3. **ItclFilter**: filtro que actualiza tres valores distintos con cada recepción de datos:

   a) **Potencia alpha**: mide las ondas alpha. Estas permiten ver si la persona se encuentra en estado de relajación. Un ejemplo de estas es

el de la figura 7, donde hay un periodo de mucha actividad de esta onda y otro de baja actividad. Esta es la visualización de un conjunto pregrabado de datos en el que la persona se relaja y cierra los ojos en la primera mitad y los abre en la segunda

b) **Workload**: como indica el nombre, es un valor que aumenta cuando estamos concentrados en la resolución de una tarea como contar descendentemente, resolver un problema matemático etc.

c) **Engagement**: muestra la respuesta del sujeto ante diferentes estímulos. Esto puede por ejemplo mostrar si un anuncio o producto llama la atención del este.

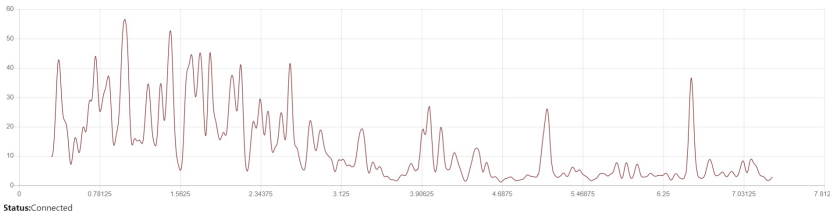

Figura 7: Captura del visor web reproduciendo la potencia alpha media de los 16 canales de un conjunto de datos pregrabados de EEG donde el sujeto se encuentra con los ojos cerrados en la primera mitad y cerrados en la segunda. (*El tiempo de la parte inferior no corresponde al real. El visor recibe a una frecuencia de 1 HZ pero la muestra como si estuviese a 256 Hz)

6. Procesamiento de Vídeo

Todo el conjunto de acciones de este proceso, captación, procesado y streaming, gira en torno a la aplicación *open-source* gstreamer. Tal y como se explica en su apartado correspondiente, este permite captar procesar y hacer streaming de vídeo de diversas fuentes mediante un sistema de tuberías. Los motivos para la selección de este se pueden observar más a fondo en el anexo A, pero las principales causas son:

1. La captación de vídeo es compatible con la cámara Realsense.
2. Usa un sistema de tuberias que da una gran libertad sobre la entrada y la salida (*source* y *sink*), permitiendo visualizar, grabar tomar instantaneas o incluso restreamear lo procesado.
3. Da soporte para la mayoría de formatos de streaming.
4. Es el software más usado en proyectos similares. Y en caso de de usar otro software, estos suelen ser herramientas de más alto nivel que integran gstreamer.
5. Hoy en día, Nvidia mejora el rendimiento de gstreamer en la Jetson debido a la cantidad de casos de uso para ella en este ámbito [16].

6.1. Despliegue

Para el caso de uso que se plantea, se debe hacer uso de la cámara simultáneamente para grabar y hacer distintos tipos de streaming. Sin embargo, solo una instancia de gstreamer puede hacer uso de la cámara en un instante de tiempo. Para esto se hace uso de el software *open-source* de RidgeRun [7, 9]:

- **Interpipe**: permite a una tubería de gstreamer establecer como fuente otra tubería de gstreamer. De esta manera, se puede establecer una que usa la cámara y que aporte el vídeo a todas las demás [9].
- **GstDaemon**: normalmente las instancias de gstreamer se lanzan como cualquier otro proceso y están constantemente operando hasta que este se interrumpen. Esta aplicación añade un estado a estas instancias, de forma que una tubería puede estar en estado *NULL*, *paused* o *playing*. De esta manera, se pueden controlar cómodamente todas las tuberías que hacen uso del vídeo de la cámara [8].

Mediante el uso de estas dos herramientas, se plantea el sistema de *pipelines* que se muestra en la figura 8. Como ejemplo de como se instancias estas tuberías, se muestra a continuación la instancia de la *Pipeline 1* de dicha figura. Se pueden ver ejemplos más básicos de como funciona gstreamer en páginas de ejemplos como la de la referencia bibliográfica [17].

```
gstd-client pipeline_create camera v4l2src device=/dev/video2
  ! video/x-raw, framerate=30/1, width=1280, height=720
  ! queue max-size-buffers=3 leaky=downstream ! nvvidconv
  ! video/x-raw ! interpipesink name=camera sync=false
```

Sobre este sistema planteado, se pueden incorporar muy fácilmente sistemas de procesado de vídeo haciendo que funcionen como fuente de las demás tuberías. Esto se puede conseguir mediante el desarrollo de plugins de gstreamer que modifiquen el buffer de entrada para confeccionar la salida deseada.

6.2. YOLO y Deepstream

Para el primer uso de la Jetson, se plantea el uso de uno de los sistemas de detección de objetos más populares, YOLOv3. Este ofrece sus redes de código abierto para que cualquiera haga uso de ellas. El principal problema es que este hace un uso muy grande de recursos, e incluso ejecutando sobre la GPU ofrece un vídeo de entre uno y tres frames por segundo. Como solución a esto, ofrece su versión más ligera YOLOv3-tiny que aumenta la tasa de frames a entre 10 y 15 por segundo a cambio de una peor detección de objetos. Se puede encontrar información de estos dos modelos y como usarlos en su página web [18]. En esta se ve como el modelo pequeño opera a 5.56 FLOPS mientras que el normal llega a casi 50 en sus versiones más ligeras.

En las actualizaciones de finales de 2019 y principios de 2020, nvidia lanzó un nuevo software, Deepstream [19]. En su página se pueden ver todas las aplicaciones que tiene y el rendimiento que tienen distintos modelos sobre cada placa de Nvidia. Resumidamente, este software facilita y acelera (tanto en tiempo de desarrollo como en eficiencia computacional) la creación de tuberías

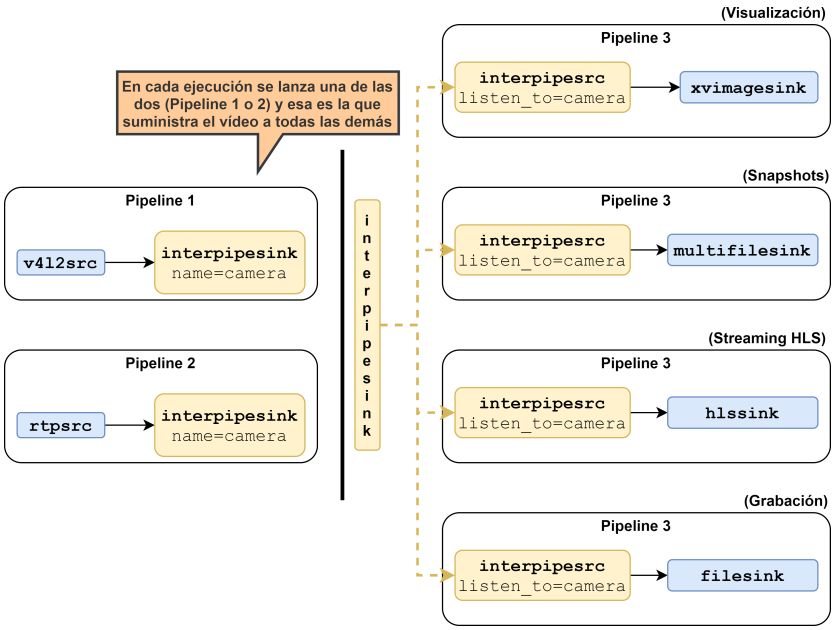

Figura 8: Despliegue del conjunto de pipelines que funcionan durante el streaming de vídeo de la Jetson

de streaming. Además, dentro de este software, ya hay desplegaos ejemplos de uso específicos de YOLO que mejoran el rendimiento del mismo YOLOv3-tiny en diez frames por segundo. Con esto sobre la mesa, definitivamente se acaba empleando el modelo de YoloV3 que se plantea en el enlace de la referencia [20]. En este modelo, se aligera el sistema con medidas como hacer streaming de mas frames de los que se procesan por completo.

	frames por segundo	**Calidad de detección**
YOLOv3	1-3	Muy alta
YOLOv3-tiny	15-20	Baja
Custom YOLOv3	10-15	Alta

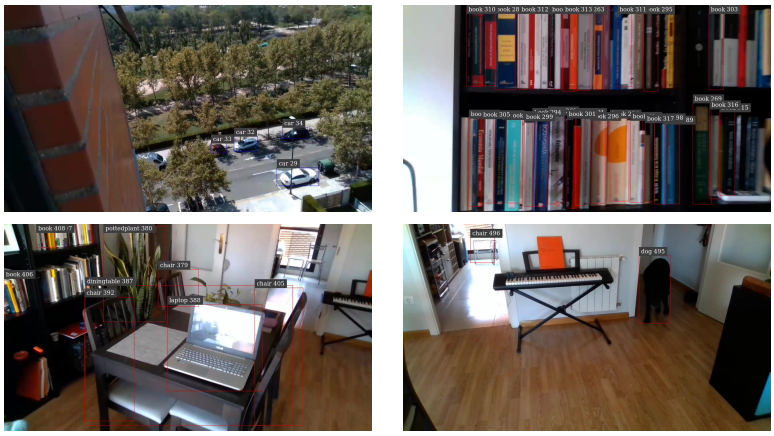

Figura 9: Fotogramas extraídos de una grabación de vídeo con la versión personalizada de YOLOv3

7. Evaluación Experimental

Esta sección muestra las pruebas realizadas sobre el prototipo final. El objetivo es valorar experimentalmente si este sirve para el propósito planteado y la eficiencia con la que lo hace. Por motivos referentes a la pandemia del COVID-19 en 2020, no se ha podido acudir a las oficinas de Bitbrain para probar el sistema final con un dispositivo de grabación de EEG. Por ello, las pruebas se realizan tomando o bien datos aleatorios del lector (representativos en cuanto a análisis, pero no permiten experimentar un caso de uso 100 % real del sistema) o bien datos pregrabados leídos de un fichero (permiten simular mínimamente casos reales). Por esto no se pueden llevar a cabo pruebas de consumo y se considera que los resultados de ambas secciones sobre procesamiento de vídeo y EEG representan el cumplimiento de los objetivos del sistema, faltando así solo una parte para valorar la eficiencia de la plataforma. Para esta última, se usan diversas herramientas para captar la saturación de los distintos componentes del sistema ante distintos escenarios.

7.1. Consumo de recursos

Para este análisis, se emplea la Jetson Nano en su modo de mayor consumo y rendimiento (modo 10W), ya que, según se ha testeado, en modos de menos consumo, no se puede ejecutar siquiera el procesamiento de vídeo, el cual necesita hacer uso del 100 % de la GPU. En este modo, se inicia el sistema sin ninguna conexión mas que la cámara y la fuente de alimentación para conseguir unos datos lo mas cercanos posible a una experiencia de uso normal, y por conexión ssh, se extrae información del sistema por medio de las siguientes dos herramientas (se puede ver el tipo de salidas interpretadas en el Anexo B):

- **Perf**: a diferencia que con las interrupciones donde se uso para analizar los eventos en cpu, aquí se usa la opción *"perf stat"* que muestra información sobre la ejecución de un programa concreto (tiempo en cpu, instrucciones por segundo etc). Esto se usa sobre los dos programas principales que corren sobre la CPU: el controlador del ADC y el filtro de EEG.

- **Tegrastats**: se trata de un software integrado en el sistema operativo instalado que una vez ejecutado, recopila información sobre la carga de la CPU y GPU, la ocupación de memoria, etc.

La tabla 2 recoge los resultados obtenidos con perf del programa de control del ADC y cada uno de los tipos de filtrado de EEG. De la información disponible se considera relevante el tiempo que pasa ejecutándose sobre la CPU, los ciclos e instrucciones ejecutados y en menos medida los fallos de acceso a cache, ya que estos últimos siguen una progresión lineal con el tiempo que el programa pasa en ejecución (cuanto mas tiempo tarda el programa en volver a ejecutarse, más de sus elementos se borran de la cache). Por un lado, sobre el filtrado de EEG, se puede ver que tanto el reenvío de los datos (sin filtro), como el filtro de impedancias consumen muy poco tiempo de CPU, lo cual lógicamente implica mas fallos en cache y ejecutan muy pocas instrucciones. Sin embargo, el filtro mas complejo, tiene una actividad en CPU de casi un 40 %, esto era previsible, al ser un filtrado con mas operaciones que procesa 16 canales, genera un vector de 121 elementos (frecuencias) y opera con el. Aunque son cifras altas que se podrían bajar con optimización, procesar un bloque de datos lleva tan solo un 40 % del tiempo disponible, lo cual no supone mucho estrés sobre un sistema con un procesador de cuatro núcleos. Por otro lado, el dato principalmente preocupante es el tiempo en CPU del controlador. No ejecuta muchas mas instrucciones que el anterior, pero se ejecuta durante más del doble de tiempo. Esto se debe a las esperas activas que se explican en su apartado y que sin duda deberían de ser el primer problema a solucionar en una siguiente iteración del proyecto, ya que se podría reducir en gran medida el estrés de la CPU.

	Tiempo en CPU(x/30s)	**Millones de ciclos**	**Millones de instrucciones**	**Instrucciones por ciclo**	**MPKI (misses per kilo instructions)**
Controlador de ADC	26,3s	37.414	19.491	0,52	0,92
Filtro EEG: sin procesado	0,94s	1.307	389	0,30	18,9
Filtro EEG: impedancias	0,96s	1.342	419	0,31	18,84
Filtro EEG: ITCL	11,8s	16.724	15.369	0,92	1,7

Cuadro 2: Medidas extraídas por medio de *"perf stat"* relevantes sobre el sistema

Por medio de tegrastats obtenemos una información más general del sistema que se ejemplifica en la figura 10. Esta representa como va aumentando el estrés del sistema conforme se despliegan procesos. Se ve claramente que aparte de los analizados con perf, el proceso que más estrés provoca tanto en CPU como GPU, es el procesamiento de vídeo por medio de Deepstream. Este es un proceso que no se puede llegar a optimizar mucho a no ser que se prueben otros reconocedores o sistemas de procesamiento. Sin embargo, el sistema en su punto de mayor estrés sigue teniendo disponible una tercera parte de la memoria y cerca de la mitad de la CPU, lo cual es un resultado más que aceptable y demuestra la capacidad de la Jetson para llevar a cabo simultáneamente un conjunto de tareas muy complejas. Aunque se ve que aun queda Memoria y CPU suficiente para ejecutar mas procesos. Añadir más puede puede llegar a ser contraproducente, ya que

más procesos concurrentes ejecutándose en la placa pueden derivar a una mala gestión del scheduler que provoque el mal comportamiento de los programas.

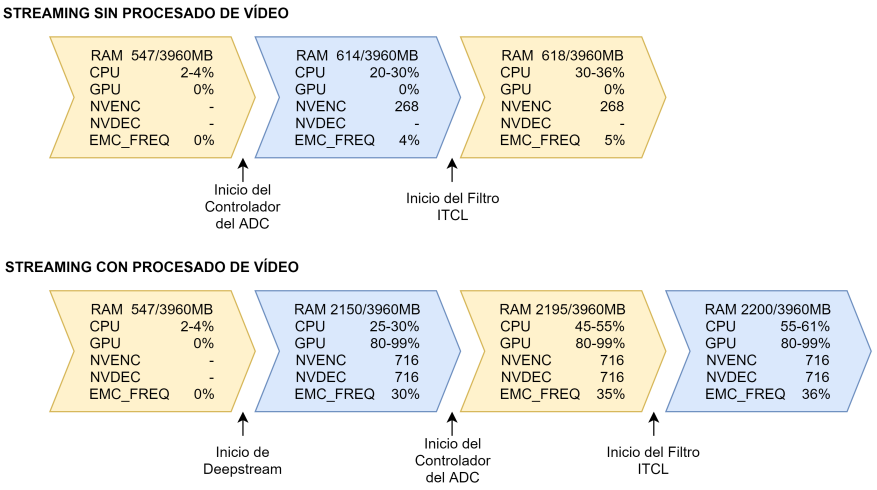

Figura 10: Representación gráfica del estrés del sistema en dos ejecuciones diferentes según la información aportada por tegrastats

8. Conclusiones

Definitivamente, se puede afirmar que se ha construido un prototipo que cumple con todas las funcionalidades planteadas en su inicio. Este es además el primero de su tipo (sistemas que hagan captación y procesado al mismo tiempo) desarrollado por Bitbrain. Así pues, en colaboración con ellos, se ha terminado un proyecto que empieza desde cero y sobre el que la empresa no tenía experiencia y referencias previas. Además, para su desarrollo se ha seguido un desarrollo vertical en el que no solo se han empleado conocimientos nuevos adquiridos durante el proyecto como las tecnologías de streaming, sino que se han puesto en práctica la mayoría de conocimientos adquiridos durante la carrera. En este proceso se han llevado a cabo desde tareas de muy bajo nivel, como el establecimiento de una comunicación entre dos piezas de hardware usando herramientas tales como un osciloscopio, hasta tareas de mucha más abstracción como la creación de un servidor web mediante librerías de python. Para cada una de las tareas de las diferentes áreas, se ha contado con la colaboración de los distintos equipos de desarrollo de Bitbrain especializados en cada una de ellas.

Con estas bases, el proyecto demuestra la viabilidad del uso de plataformas como la Jetson para este tipo de prototipos, que tal y como se ve en los resultados cumple con los requisitos computacionales. Por otro lado, al ser un proyecto tan amplio, el limitado tiempo de desarrollo de un TFG ha hecho que se queden algunos frentes abiertos de desarrollo. Algunos de estos son temas bastante necesarios de abordar en caso de continuar con el desarrollo, mientras que otros son plantillas que se dejan abiertas para el desarrollo de nuevos elementos, como la implementación de diferentes tipos de procesado de vídeo y EEG. Estos se plantean en el siguiente subapartado.

8.1. Trabajo futuro

Se ordenan en orden de prioridad el trabajo que se considera que sería necesario en caso de considerar mas horas de desarrollo del proyecto.

- Repetir una investigación sobre los distintos tipos de interrupciones una vez aplicado el parche en tiempo real para ahorrar recursos de CPU.
- comprobar y asegurar que los procesados de EEG tienen suficientes recursos como para no saltarse ni una sola recepción de datos y mantener un funcionamiento a 256Hz.
- Mejorar la seguridad controlando los accesos a los recursos de root. El GPIO se tiene que manipular como root, sería necesario buscar la manera de dar acceso a todo los usuarios a los pines usados para no tener que ejecutar nada con tanta autoridad.
- Incorporar un detector de eventos relacionados con timestaps a partir de los objetos detectados en el procesamiento de vídeo.
- Estudiar el código javascript de la gráfica de la web para una mejor visualización del EEG.
- Investigar más métodos de streaming alternativos para evitar el delay del HLS.

Referencias

[1] "Página principal del producto de grabación de eeg dreem." https://dreem.com/.

[2] "Página principal del producto de grabación de eeg focuscalm." https://www.focuscalm.com/.

[3] Nvidia, "Guia oficial de arranque de la jetson nano." https://https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit#write.

[4] "Real-time linux wiki." https://rt.wiki.kernel.org/index.php/Main_Page.

[5] "Github del proyecto librealsense." https://github.com/IntelRealSense/librealsense.

[6] "Página principal de gstreamer." https://gstreamer.freedesktop.org/.

[7] "Página principal de ridgerun." https://www.ridgerun.com/.

[8] "Framework de ridgerun gstdaemon." https://developer.ridgerun.com/wiki/index.php?title=GStreamer_Daemon.

[9] "Proyecto de github del framework de ridgerun interpipe." https://github.com/RidgeRun/gst-interpipe.

[10] "Librería en c++ para la counicación por websockets." https://github.com/zaphoyd/websocketpp.

[11] "Librería en c++ para el uso de elementos json." https://github.com/nlohmann/json.

[12] "Documentación de spidev." https://www.kernel.org/doc/Documentation/spi/spidev.

[13] "Explicación de la comunicación por spi." https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/all.

[14] "Proyecto de github de traceshark." https://github.com/cunctator/traceshark.

[15] "Proyecto de github de hls.js." https://github.com/video-dev/hls.js/.

[16] "Gstreamer acelerado po nvidia." https://docs.nvidia.com/jetson/archives/l4t-archived/l4t-3231/index.html#page/Tegra%20Linux%20Driver%20Package%20Development%20Guide/accelerated_gstreamer.html.

[17] "Ejemplos sencillos de uso de gstreamer." http://wiki.oz9aec.net/index.php/Gstreamer_cheat_sheet.

[18] "Página principal de yolo." https://pjreddie.com/darknet/yolo/.

[19] "Página principal de deepstream." https://developer.nvidia.com/deepstream-sdk.

[20] “Entrada del foro de nvidia que plantea una instancia de yolov3 que funciona a 20fps.” https://forums.developer.nvidia.com/t/deepstream-gst-nvstreammux-change-width-and-height-doesnt-affect-fps/83286.

A. Streaming de Vídeo

Uno de los principales problemas afrontados del proyecto es la forma de tratar, enviar y recibir vídeo. En este anexo se resumen las tecnologías y formatos que se han probado para implementar dicha funcionalidad y el porqué de las elegidas finalmente.

A.1. Tecnologías de Streaming

En primer lugar, se necesita un sistema que nos permita realizar un streaming de vídeo desde la cámara Intel Realsense. Para esto se plantean las siguientes opciones:

- ROS (Robot Operating System): uno de los frameworks más usados para la creación de software para robots. Además, se plantea el uso de la parte de streaming de vídeo de otros protocolos de robótica como *Mavlink* o *Motion*.
- Gstreamer: software de freedesktop incorporado en Linux para el streaming de vídeo, audio y datos.

A.1.1. Frameworks de robótica VS Gstreamer

Lo primero que se ha probado ha sido el software ofrecido por mavlink. Este tiene un protocolo de comunicación para cámaras basado en un stream rtsp que simplifica la detección de la cámara y la emisión mediante archivos de configuración con su respectivo formato. Sin embargo, finalmente, la emisión de este stream se lleva a cabo por medio de Gstreamer. La decisión queda pues al nivel de abstracción que se desee. Por eso, sería preferente usar Gstreamer para un mayor control sobre el sistema.

A.2. Formatos de streaming

Los posibles formatos de streaming de vídeo que podrían plantearse son: RTSP, RTMP y HLS. Todos ellos usan una compresión de video H264, el método más utilizado debido a la calidad que ofrece junto a una gran tasa de compresión.

De estos 3 se puede descartar el RTMP, ya que es el único formato de streaming que no se puede emitir por medio de la tecnología seleccionada para el stream (Gstreamer).

A.2.1. RTSP (Real Time Streaming Protocol)

El principal problema que surge con este protocolo es que no tiene soporte por parte de HTML5. Esto provoca que sea complicado ver el contenido del stream en un servidor web, lo cual entra en conflicto con uno de los principales requisitos del sistema.

Tramnsmisión desde la Placa: para poder visualizar en una página web este formato de estreaming, al no poder dejar este trabajo al navegador, se recibe y se procesa el streaming desde el propio servidor web. Para esto se usan las herramientas de OpenCV para la reproducción de streaming en python.

Al hacer esto, para lograr ver un streaming con una latencia mínima en torno a los 0,5 segundos, la calidad del vídeo que se puede recibir son como máximo 15 frames por segundo y 640 x 480 pixels por frame.

Este método además, consume una gran cantidad de CPU de la placa, que se podría ahorrar o utilizar para otras tareas.

HTTP: se han tratado solucionar los problemas planteados por el formato anterior partiendo de que el único formato que acepta HTML5 para reproducción de video es http.

Este streaming no puede ser directo desde la cámara, ya que limitaría el trabajo con esta (La cámara solo puede ser usada por un programa al mismo tiempo). Así pues, se intenta captar, convertir y emitir en otro puerto el stream RTSP. Para esto se emplean herramientas como VLC y FFMPEG, especializadas es este tipo de conversiones de vídeo.

Se ha probado esta medida con streaming de varias calidades, y al igual que pasaba con OpenCV, la taréa de procesar el vídeo en H264 es demasiado pesada para la placa.

Soluciones: por un lado, se podría procesar el vídeo y transmitirlo usando parte de la GPU disponible. Esto permitiría hacer el streaming, pero a cambio consumiría recursos muy valiosos para el procesado de vídeo con tal de alcanzar un objetivo de menor relevancia.

Se puede hacer la conversión y retransmisión desde oro equipo de la misma red o alquilando alguno de los servicios contratables de internet para esta tarea, lo cual no encaja con los objetivos del proyecto.

Existen sistemas específicos de navegadores que podrían soportar de alguna manera el stream, como podría ser Flash Player, WebRTC o el uso de WebSockets.

A.2.2. HLS

Este es un formato de streaming funciona mediante la creación de archivos en el propio sistema de ficheros de la placa. Se crea un archivo de extensión *m3u8* y una lista finita de ficheros de vídeo. Este archivo principal, contiene referencias al los archivos de vídeo y cual se debe reproducir en cada instante de tiempo.

Además, este formato se puede emitir por medio de lo "bad plugins"de gstreamer y reproducir facilmente en la web con javascript por medio de *hls.js*, lo cual no supone apenas carga para la placa.

B. Modelo de salidas de perf y tegrastats

Se muestran aquí los formatos de salida de información de pef y tegrastats que se han interpretado para llevar a cabo el análisis del sistema.

- **perf**:

```
 Performance counter stats for './EEGfilter no_filter':

        941,829906      task-clock (msec)         #
0,031 CPUs utilized
             8.094      context-switches          #
0,009 M/sec
               715      cpu-migrations            #
0,759 K/sec
               145      page-faults               #
0,154 K/sec
     1.307.980.529      cycles                    #
1,389 GHz
       389.990.560      instructions              #
0,30  insn per cycle
   <not supported>      branches
         8.294.722      branch-misses
       170.654.674      L1-dcache-loads           #
181,195 M/sec
         7.377.223      L1-dcache-load-misses     #
4,32% of all L1-dcache hits
   <not supported>      LLC-loads
   <not supported>      LLC-load-misses
      30,074581344 seconds time elapsed
```

- **tegrastats**:

```
RAM 547/3960MB (lfb 638x4MB) SWAP 0/1980MB (cached 0MB)
IRAM 0/252kB(lfb 252kB) CPU [6%@1479,7%@1479,7%@1479,6%@1479]
EMC_FREQ 0%@1600 GR3D_FREQ 0%@76 APE 25 PLL@28.5C CPU@29.5C
iwlwifi@34C PMIC@100C GPU@30C AO@39C thermal@29.75C POM_5V_IN
1712/1712 POM_5V_GPU 0/0 POM_5V_CPU 244/244
```

C. Análisis de restricciones temporales con Perf y Traceshark

Si no se usa un sistema operativo en tiempo real para la lectura de señal EEG, empiezan a surgir unas anomalias temporales debido al mal scheduling del kernel que se presentan a continuación. Hay más casos de los que aquí se presentan como prueba, pero todos se deben a la falta de prioridad de las interrupciones para conseguir recursos.

Para estas pruebas se usa el programa Traceshark [14], que permite visualizar los datos capturados por medio de perf que muestran la actividad de la cpu durante

el funcionamiento del programa. Este visualizador permite ver entre otras cosas, la actividad de la cpu y los cambios de contexto y ver aislado el tiempo asignado a un proceso concreto. En las pruebas se muesta esta información y en una linea verde el tratamiento de la interrupción generada por el ADC. En la figura 11 se puede observar como a una constancia de 1/256 segundos se trata la interrupción en muy poco tiempo. Sin embargo, en varias ocasiones, esta falta de prioridad hace que se produzcan casos como los de la figura 12, donde salta la interrupción y no se atiende a tiempo, e incluso otros como el de la figura 13, donde se interrumpe la ejecución del tratamiento para dejar la cpu para ortros procesos con mayor prioridad.

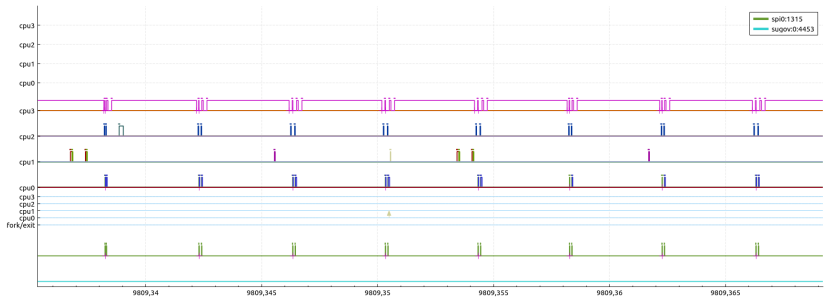

Figura 11: Traza de ejecución que muestra el schedulling normal durante la ejecución del controlador del ADC

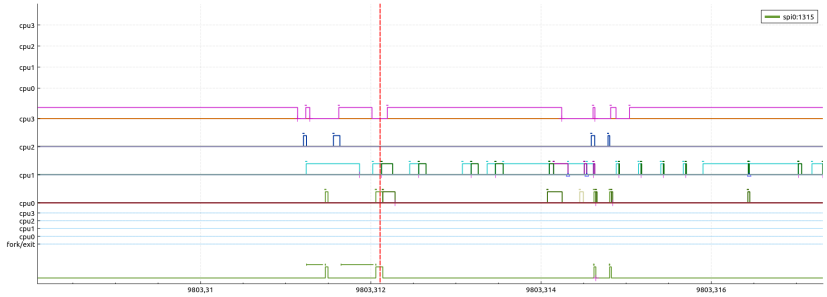

Figura 12: Traza en la que no solo se muestra el retraso a la hora de atender la interrupción, sino que también se tarda mas en ejecutar el tratamiento

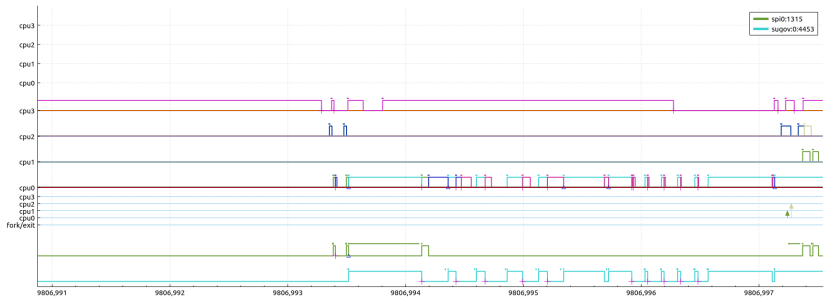

Figura 13: Traza en la que se interrumpe el tratamiento de la interrupción para ceder la cpu a otro proceso

D. Uso del GPIO

Jetson Nano J41 Header

Sysfs GPIO	Name	Pin	Pin	Name	Sysfs GPIO
	3.3 VDC *Power*	1	2	5.0 VDC *Power*	
	I2C_2_SDA *I2C Bus 1*	3	4	5.0 VDC *Power*	
	I2C_2_SCL *I2C Bus 1*	5	6	GND	
gpio216	AUDIO_MCLK	7	8	UART_2_TX */dev/ttyTHS1*	
	GND	9	10	UART_2_RX */dev/ttyTHS1*	
gpio50	UART_2_RTS	11	12	I2S_4_SCLK	gpio79
gpio14	SPI_2_SCK	13	14	GND	
gpio194	LCD_TE	15	16	SPI_2_CS1	gpio232
	3.3 VDC *Power*	17	18	SPI_2_CS0	gpio15
gpio16	SPI_1_MOSI	19	20	GND	
gpio17	SPI_1_MISO	21	22	SPI_2_MISO	gpio13
gpio18	SPI_1_SCK	23	24	SPI_1_CS0	gpio19
	GND	25	26	SPI_1_CS1	gpio20
	I2C_1_SDA *I2C Bus 0*	27	28	I2C_1_SCL *I2C Bus 0*	
gpio149	CAM_AF_EN	29	30	GND	
gpio200	GPIO_PZ0	31	32	LCD_BL_PWM	gpio168
gpio38	GPIO_PE6	33	34	GND	
gpio76	I2S_4_LRCK	35	36	UART_2_CTS	gpio51
gpio12	SPI_2_MOSI	37	38	I2S_4_SDIN	gpio77
	GND	39	40	I2S_4_SDOUT	gpio78

Figura 14: Mapa de la cabecera de pines j41 de una Jetson nano. Indicados en rojo se encuentran los pines usados y su función